

Building a CDR analytics platform for real-time services

Chia-Chun Shih, Chi-Chang Huang, Bo-Ting Lin, Chao-Wen Huang, Wan-Hsun Hu, Chien-Wei Cheng

Billing Information Laboratory
Chunghwa Telecom Laboratories
Taipei, Taiwan

{ccshih, hunter0513, btlin1025, huangcw, ringo0601, cwcheng}@cht.com.tw

Abstract—Growing real-time services require discovering transient customer needs in CDRs. In this paper, we propose a CDR analytics system architecture for event-based analytics processing in real time. This system is expected to allow marketing staff to inject analytics-based rules to trigger marketing campaigns. We also propose two implementations which are based on off-the-shelf computing tools.

Keywords—Real-time CDR processing, Real-time analytics, Stream computing, In-memory data grid

I. INTRODUCTION

Growing real-time services require discovering transient customer needs in CDRs. Customer needs are so transient that telcos should address them as quickly as possible. ETL tools, which require full dataset available before starting analysis, are insufficient for real-time services.

Imagine a scenario that a telecom system can recognize subscribers who are steadily watching high-quality streaming videos by a congested base station which is nearby a baseball stadium. These subscribers can immediately receive a campaign message which is specialized for subscribers who consume high bandwidth in congested networks. To realize this scenario, the telecom system shall be able to collect real-time analytics of streaming service consumption in dimension of subscribers and base stations. The system shall also allow marketing staffs to inject rules to trigger campaign promotions. An example workflow of the scenario is depicted in Figure 1, where a CDR analytics platform drives real-time responses for customer needs.

Besides individual behavior patterns, group behavior patterns are also discoverable from CDRs. In another scenario that a telecom system can recognize unusual video streaming usage by a base station which is nearby a baseball stadium. Spectators are likely to use game companion apps to watch instant replay, which incurs an recognizable usage patterns of sport activities. These group patterns reveal subscriber scenarios and are useful for targeted promotion or QoS control.

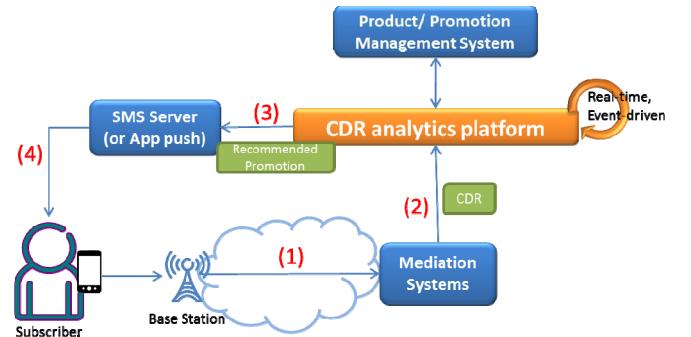


Fig. 1. An example workflow of real-time campaign promotion

This paper proposes a CDR analytics platform for real-time event-based analytics processing. This system is expected to allow marketing staffs to inject analytics-based rules to trigger actionable marketing events. This paper also proposes two implementations which are based on off-the-shelf computing tools.

Challenges of this platform are as follows.

- High volume: This system adopts existing big data utilities to handle high volume traffic. Noted that big data utilities trade generality for scalability, so it is important to carefully compare/contrast these utilities before adoption.
- Real-time processing: Analytics should be updated in seconds.
- Sophisticated event detection: Event may last in a time window or until specific condition happens.

The remainder of this paper is organized as follows. Section 2 reviews related works. Section 3 presents requirements and architecture. Section 4 proposes two implementations. Section 5 presents concluding remarks, as well as future directions.

II. RELATED WORKS

A. Real-time processing for CDRs

Some prior works aim to process CDR in (near) real time. Kar *et al.* [2] implemented a real-time CDR processing system

based on IBM Infosphere Streams [9]. This system processes stored data in offline mode and analyzes data stream in real-time for actionable intelligence. This work demonstrates benefits of applying stream computing to time-window based operation. Jayawardhana *et al.* [4] proposed a CDR analyzer for near real time telecom promotions. This work adopts Cassandra [23], a NoSQL database with high write throughput, as data store. Theeten *et al.* [5] proposed a parallel streaming engine for telco applications. This work tests diverse parameter configuration of Storm framework [6] and shows that optimal parameter for telecom system is non-trivial.

B. Online Charging Sysyem

3GPP brings up Policy and Charging Control (PCC) architecture [10] [11] [12] for real-time charging and event control in mobile 4G network. Telcos can predefine policies at Subscriber Profile Repository (SPR) of Policy and Charging Rules Function (PCRF). When conditions, such as running out of data volume, are matched, PCRF shall make decision based on policies and inform Policy and Charging Enforcement Function (PCEF) to execute designated actions, e.g., to advice of charge, QoS adjustment and to send discount information to customer.

Although it is convenient for telcos to alter business models based on PCC architecture, Online Charging System (OCS) can only keep personal charging information, such as data usage or dedicate information (time of day). However, PCC architecture does not keep history records and does not support group behavior analysis. Flexibility of policy depends on customization of OCS.

C. In-Memory Data Grid

Recently, in-memory computing emerges as a new paradigm for high performance enterprise applications [16]. In contrast to other computing paradigm, in-memory computing resides data in main memory, which is significantly faster than other storage devices.

In-Memory Data Grid (IMDG) is a realization of in-memory computing. IMDG can aggregate memory of multiple nodes as a memory pool for storing data [17]. Data is distributed across nodes. If necessary, data is replicated to multiple nodes for better availability. IMDG can also horizontally scale out.

IMDG is not only a data store, but also a data processor. IMDG provides distributed computing frameworks to distribute computing to the nodes where data resides. Data locality reduce data transmission and is beneficial for data-intensive applications. Moreover, distributing computing over multiple nodes also help achieve parallel processing.

IMDG allows applications to intercept events for customized processing. A typical scenario is to intercept CRUD events to trigger another CRUD events. Interception can also act as triggers to glue multiple processing steps.

Many off-the-shelf IMDG products are available, e.g., VMWare Gemfire [18], Oracle Coherence [19], Gigaspaces XAP [20], Hazelcast [21] and Red Hat Jboss data grid [22].

D. Stream Computing

Stream computing is a data processing model for real-time/near-real-time applications. A stream is a sequence of infinite data elements which continuously arrive over time. Stream computing connects data streams and data processors in pipeline. It is still a research topic to develop scalable, elastic and maintainable stream computing models [13] [14].

A stream computing system is composed of multiple nodes, each host multiple processing units. A processing unit represents a step in a computing process, such as filter or translation. Based on a pre-defined processing topology, data flows through processing units. Stream computing keeps traceability of data, which is the basis of fault-tolerance. In summary, stream computing is a real-time computing paradigm for high availability and high performance.

Many stream computing tools have been developed, e.g., Apache Storm [6] [15] originated from Twitter, S4 [7] developed by Yahoo, Apache Samza [8] originated from LinkedIn and InfoSphere Streams [9] developed by IBM.

III. PROPOSED ARCHITECTURE

This section proposes requirements and architecture of this system.

We plan to extend a SQL-based CDR query system to support analytics. We expect the new system to support the depicted scenarios in Section 1. Requirements include:

- Real-time analytics: The platform shall be able to manage about billions of metrics. Traditional data warehousing system can process billions of analytics in batch, but hardly in real-time.
- Alleviate RDB overhead: Current CDR query system is based on a custom sharded relational database (RDB) which is dedicatedly serving CDR queries. It is doubted to implement analytics and events functionalities on RDB without significant performance loss. Furthermore, although RDB is capable of triggering events, it is inefficient to handle thousands of triggers.
- Scalable architecture: The system shall handle large number of metrics. It is beneficial to adopt distributed architecture for high scalability.
- Handle complex and delayed events: A single event may consist of multiple time-dependent conditions, which may be triggered in a time window. For example, an event “when a user enters a specific area and stays for at least 30 minutes” is composed of three time-dependent conditions: (1) Time 0 min: A user walks into a specific area, (2) Time 0 to 30 min: This user is not out of this area, and (3) Time >30 min: This user is in this area. The three conditions span a time window and are difficult to check simultaneously.

Our architecture is based on Lambda architecture [3]. Lambda architecture is a data processing architecture aimed to satisfy the needs for a scalable, fault-tolerant, and robust system to serve a wide range of workloads and use cases

which requires low-latency access. Lambda architecture is composed of three layers: batch layer, serving layer and speed layer. Incoming data is fed into both batch layer and speed layer. Batch layer manages master dataset and pre-compute batch views. Serving layer complements ad-hoc query capability, which is missing in batch layer. It is time-consuming for new data to be available in service layer because index and preprocessing takes time. The speed layer compensates for the high update latency of serving layer, but only process recent data. This paper mostly focuses on speed layer.

Figure 2 shows the proposed architecture based on Lambda architecture. Batch layer contains two databases -- One for CDRs, and the other for customer data. Serving layer serves CDR queries. This paper focuses on speed layer.

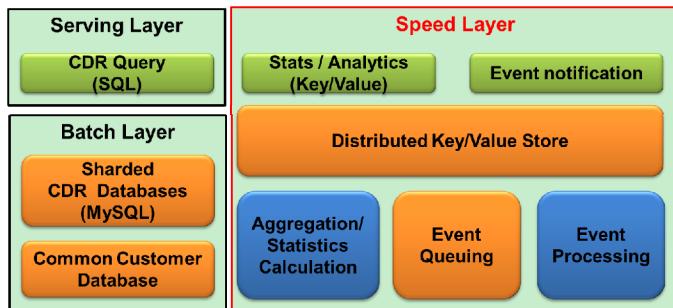


Fig. 2. Proposed architciture for CDR analytics platform

The proposed speed layer is composed of the following main components:

- A key/value data store for managing analytics
- An efficient computing component to calculate analytics in real time
- A event processing component allows time-dependent events and real-time event detection
- A reliable messaging system to glue components coherently (if necessary)

The following subsections introduce responsibilities of these main components.

A. Key-Value Data Store

The key-value data store keeps analytics. Keys can be combinations of the following attributes:

- Subscriber: may be an individual subscriber, a group subscriber or an arbitrarily defined subscriber group
- Time interval or time window: a static time interval with different granularity or a time window
- Location: locations are identified by one or more cell ids
- Behavior: used volume, requested value added services and so on
- Complex events: events preconditioned by other events

For better scalability, it is desirable to adopt distributed key-value data stores. However, distributed key/value stores may suffer from high network latency and high network bandwidth consumption.

This key-value data store shall afford thousands of concurrent access with low latency.

It is desirable to adopt rolling-up operations to reduce number of analytics, however, it is questionable to efficiently roll up thousands, or even millions of analytics at runtime.

The key-value data store shall be reliable and fault-tolerant.

B. Analytics Calculation and Event Processing

This system shall calculate analytics in real time. If operationally affordable, it is also desirable to adopt event-based method to handle incoming data stream.

This system shall calculate time-window statistics. Complex analytics, which is derived from machine learning algorithms, are not covered in this paper.

C. Reliable Message Queue

Contemporary distributed systems use message queues to instantly and reliably exchange messages in real-time. This system prefers distributed message queues for high concurrency and high throughput. Message replay is also desirable for better fault tolerance.

IV. IMPLEMENTATION OPTIONS

This section introduces two implementation options. The first implementation option is based on stream computing and the other is based on IMDG.

A. Stream Computing Implementation

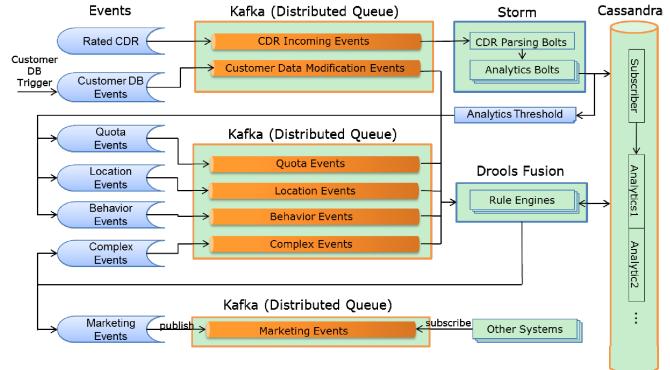


Fig. 3. Stream Computing Implementation

Figure 3 illustrates steam computing implementation. This implementation combines four utilities:

- Apache Storm [6] for analytics calculation: Storm is a horizontally scalable framework for real-time processing of streaming data. Storm encodes data processing logic as topologies, which can be further decomposed into spouts and bolts. Spouts create streamlined data elements, which are called tuples. Bolts, which process tuples and emit new tuples to one

or multiple bolts for succeeding processing. Each bolt may be composed of multiple workers for parallel processing. Workers can be distributed across nodes for horizontal scalability.

- Cassandra [23] as data store: Cassandra is a scalable and high-performance distributed key-value store. Cassandra can achieve extraordinary write throughput and is an excellent building block of this system. Cassandra stores data by column, which is a good-fit for analytics application. Cassandra adopts consistent hashing algorithm to distribute data. Consistent hashing algorithm eliminates overhead of dynamic scaling, however, data locality may be lost because consistent hashing algorithm is inflexible to customize. A remedy is to align computing nodes for better data locality. For example, Storm supports fields grouping, which fixes assignments of tuples to workers, and pluggable scheduler, which fixed assignments of workers to nodes. Fixed mappings between tuples and nodes leads to better data locality.
- Drools Fusion [24] as event processing tool: Drools Fusion is a tool for Complex Event Processing (CEP), which can process event stream based on time-dependent rules. Drools Fusion also provides a domain-specific language for authoring rules.
- Apache Kafka [25] as message queue: Kafka is a distributed publish-subscribe message queue. Kafka can efficiently persist data and afford to keep TB's of data for data replay. Kafka divide messages into topics, which can further divide into partitions with replication. A report [1] shows Kafka can captures 55 billion events per day, which should be sufficient for our use. This system adopts Kafka to store and dispatch event messages.

Figure 3 shows workflow of this implementation. Events from CDR and customer DB updates are sent to separate topic channel of Kafka. CDR parsing bolts are implemented for different types of CDRs. Parsed CDRs are sent to analytic bolts, which update analytics in Cassandra. If the updated analytics exceed predefined thresholds, analytic bolts will emit event messages (e.g., quota events, location events, or user-defined behavior events) to Kafka. Rule engines, which are based on Drools Fusion, are listening events in order to decide whether to emit a marketing event message. Marketing event messages are sent to another Kafka topic channel for consumption by other systems.

Note that Storm is stateless computing model and does not support recovery. To solve the problem, this implementation defers all state updates to last bolts of topology. Intermediate bolts only emit update requests to downstream bolts and does not update analytics. Because all updates occurs at a single bolt of topology, it is easier to recover.

This implementation combines strengths of off-the-shelf tools but may lack manageability. Next subsection introduces another implementation based on IMDG.

B. In-Memory Data Grid (IMDG) Implementation

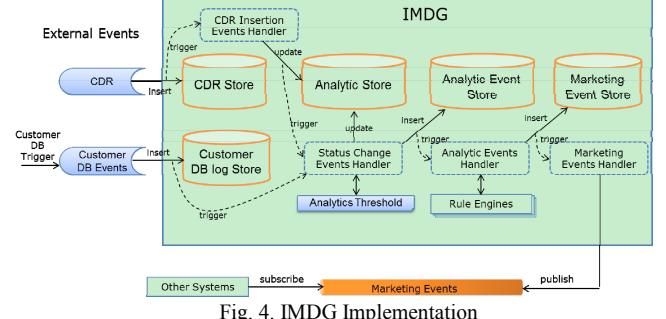


Fig. 4. IMDG Implementation

Figure 4 illustrates IMDG-based implementation. In this implementation, IMDG serves multiple functionalities, including data stores, analytics calculators and event handler.

This implementation hosts five separate data stores in IMDG.

- CDR Store saves incoming CDRs.
- Customer DB log stores modification events triggered by customer database.
- Analytic Store saves calculated analytics. It is the only data store which requires persistence for failure recovery.
- Analytic Event Store saves events which are triggered when analytics exceed thresholds.
- Marketing Event Store saves marketing campaign messages triggered by rule engines.

Most data stores only serve as transient stores for triggering events. Data entries are disposable once events are triggered. Therefore it is usually sufficient to use non-persistent memory-based storage. Moreover, this system requires efficient event processing, which can benefit from in-memory processing.

Because IMDG supports data processing frameworks, it is unnecessary to adopt another computing framework, e.g. Storm.

Unlike stream computing implementation, which adopts Kafka queue to reliably deliver messages between components, IMDG keeps most communication inside a platform so that it only requires a queue to deliver marketing events to other systems.

This implementation heavily relies on event trigger mechanism to glue multiple processing steps. However, it is doubtful for IMDG to handle millions of concurrent events with low latency.

C. Discussions

This subsection discusses pros/cons of the two implementations.

The stream computing implementation incorporates multiple tools, in which each tool concentrates on a specific task. This implementation has the following advantages.

- High flexibility: It is easy to alter any tool for specific requirements. Moreover, incorporation of diverse tools offers rich configurable options for different scenarios.
- Meeting demanding requirements: Specialized tools are more likely to meet demanding requirements than all-in-one tools.

The IMDG implementation delegates most functionalities to IMDG. The implementation has the following advantages.

- Better data locality: Data locality impacts performance especially when updates occur frequently. Our preliminary experiments show 70%-80% throughput degradation when half of data is remotely located. IMDG, which coherently handles data and computation, has better data locality.
- Better manageability: Keeping data and computation in a single box is more manageable. Manageability is especially important for transaction-oriented event processing.

It still requires further exploration to finalize our decision among the two implementation choices.

V. CONCLUDING REMARKS AND FUTURE DIRECTIONS

Growing real-time services require discovering transient customer needs in CDRs. In this paper, we propose a CDR analytics system architecture for large-scale event-based analytics processing in real time. This system allows marketing staffs to inject analytics-based rules to trigger marketing campaigns. We also propose and discuss two implementations which are based on off-the-shelf tools.

Some analytics management issues are still untouched in this paper:

- Analytics affinity: Some analytics are highly correlated. It is possible to group calculations of these analytics for better performance.
- Analytics invalidation: Rule changes should accompany with analytics invalidation. A reference table shall be kept for invalidating unused analytics.

In the future, we plan to integrate more input sources other than CDRs to enrich application scenarios. We also plan to measure performances of the two implementations under high concurrency workload.

REFERENCES

- [1] K. Goodhope, J. Koshy, J. Kreps, N. Narkhede, R. Park, J. Rao and V. Y. Ye, "Building LinkedIn's Real-time Activity Data Pipeline," *IEEE Data Engineering Bulletin*, Vol.35 no.2, pp.33-45, 2012.
- [2] D. Kar, P. Misra, P. Bhattacharjee, and A. Mukherjee, "Real-time telecom revenue assurance," *Proceedings of the Seventh International Conference on Digital Telecommunications*, pp.130-135, 2012.
- [3] N. Marz and J. Warren, "Big Data: Principles and best practices of scalable realtime data systems," Manning Publications, 2013.
- [4] P. Jayawardhana, A. Kumara, D. Perera and A. Paranawithana, "Kanthaka: Big Data Caller Detail Record (CDR) Analyzer for Near Real Time Telecom Promotions," *Proceedings of the Fourth International Conference on Intelligent Systems Modelling & Simulation (ISMS)*, pp.534-538, 2013.
- [5] B. Theeten, I. Bedini, P. Cogan, A. Sala and T. Cucinotta, "Towards the Optimization of a Parallel Streaming Engine for Telco Applications," *Bell Labs Technical Journal*, vol.18 no. 4, pp. 181–197, 2014.
- [6] Apache Storm, <http://storm-project.net>.
- [7] L. Neumeyer, B. Robbins, A. Nair and A. Kesari, "S4: Distributed stream computing platform", *Proceedings of IEEE International Conference on Data Mining Workshops (ICDMW)*, pp. 170-177, 2010.
- [8] Apache Samza, <http://samza.incubator.apache.org>.
- [9] C. Ballard, D. M. Farrell, M. Lee, P. D. Stone, S. Thibault and S. Tucker, "IBM InfoSphere Streams Harnessing Data in Motion," *IBM Redbooks*, 2010.
- [10] 3GPP, "3GPP TS 23.203 Policy and charging control architecture (Release12)," Retrieved 2014.
- [11] 3GPP, "3GPP TS 29.219 Policy and Charging Control: Spending Limit Reporting over Sy reference point (Release12)," Retrieved 2013.
- [12] 3GPP, "3GPP TS 29.213 Policy and Charging Control signalling flows and Quality of Service(QoS) parameter mapping (Release12)," Retrieved 2014.
- [13] V. Gulisano, R. Jimenez-Peris, M. Patino-Martinez, C. Soriente and P. Valduriez, "StreamCloud: A Large Scale Data Streaming System," *IEEE Transactions on Parallel and Distributed Systems*, vol.23 no. 12, pp. 2351-2365, 2012.
- [14] A. Ishii, and T. Suzumura, "Elastic Stream Computing with Clouds," *Proceedings of IEEE International Conference on Cloud Computing (CLOUD)*, pp. 195-202, 2011.
- [15] M. T. Jones, "Process real-time big data with twitter storm," *IBM Technical Library*, 2013.
- [16] H. Plattner and A. Zeier, "In-memory data management: an inflection point for enterprise applications," *Springer*, 2011
- [17] W. Bain and M. Sobolev, "How In-Memory Data Grids Can Analyze Fast-Changing Data in Real Time," *Scale-out Software*, 2012. (available at http://www.odbms.org/wp-content/uploads/2013/11/Analysis_Fast-Changing.pdf)
- [18] VMWare Gemfire, <http://www.vmware.com/products/vfabric-gemfire>.
- [19] Oracle Coherence, <http://www.oracle.com/technetwork/middleware/coherence/overview/index.html>.
- [20] Gigaspaces XAP, <http://www.gigaspaces.com/xap-in-memory-computing-event-processing/Meet-XAP>.
- [21] Hazelcast, <http://hazelcast.org/>.
- [22] Red Hat JBoss data grid, <http://www.redhat.com/products/jbosserverprisemiddleware/data-grid/>.
- [23] Cassandra, <http://cassandra.apache.org/>.
- [24] Drools Fusion, <http://drools.jboss.org/drools-fusion>
- [25] Apache Kafka, <http://kafka.apache.org/>